

How Unsplittable-Flow-Covering helps Scheduling with Job-Dependent Cost Functions^{*}

Wiebke Höhn¹, Julián Mestre², and Andreas Wiese³

¹ Technische Universität Berlin, Germany. hoehn@math.tu-berlin.de

² The University of Sydney, Australia. mestre@it.usyd.edu.au

³ Max-Planck-Institut für Informatik, Saarbücken, Germany.
awiese@mpi-inf.mpg.de

Abstract. Generalizing many well-known and natural scheduling problems, scheduling with job-specific cost functions has gained a lot of attention recently. In this setting, each job incurs a cost depending on its completion time, given by a private cost function, and one seeks to schedule the jobs to minimize the total sum of these costs. The framework captures many important scheduling objectives such as weighted flow time or weighted tardiness. Still, the general case as well as the mentioned special cases are far from being very well understood yet, even for only one machine. Aiming for better general understanding of this problem, in this paper we focus on the case of uniform job release dates on one machine for which the state of the art is a 4-approximation algorithm. This is true even for a special case that is equivalent to the covering version of the well-studied and prominent unsplittable flow on a path problem, which is interesting in its own right. For that covering problem, we present a quasi-polynomial time $(1+\varepsilon)$ -approximation algorithm that yields an $(e+\varepsilon)$ -approximation for the above scheduling problem. Moreover, for the latter we devise the best possible resource augmentation result regarding speed: a polynomial time algorithm which computes a solution with *optimal* cost at $1+\varepsilon$ speedup. Finally, we present an elegant QPTAS for the special case where the cost functions of the jobs fall into at most $\log n$ many classes. This algorithm allows the jobs even to have up to $\log n$ many distinct release dates.

1 Introduction

In scheduling, a natural way to evaluate the quality of a computed solution is to assign a cost to each job which depends on its completion time. The goal is then to minimize the sum of these costs. The function describing this dependence may be completely different for each job. There are many well-studied and important scheduling objectives which can be cast in this framework. Some of them are already very well understood, for instance weighted sum of completion times $\sum_j w_j C_j$ for which there are polynomial time approximation schemes (PTASs) [1], even for multiple machines and very general machine models. On

^{*} Funded by the Go8-DAAD joint research cooperation scheme.

the other hand, for natural and important objectives such as weighted flow time or weighted tardiness, not even a constant factor polynomial time approximation algorithm is known, even on a single machine. In a recent break-through result, Bansal and Pruhs presented a $O(\log \log P)$ -approximation algorithm [7,6] for the single machine case where every job has its private cost function. Formally, they study the General Scheduling Problem (GSP) where the input consists of a set of jobs J where each job $j \in J$ is specified by a processing time p_j , a release date r_j , and a non-decreasing cost function f_j , and the goal is to compute a preemptive schedule on one machine which minimizes $\sum_j f_j(C_j)$ where C_j denotes the completion time of job j in the computed schedule. Interestingly, even though this problem is very general, subsuming all the objectives listed above, the best known complexity result for it is only strong NP-hardness, so there might even be a polynomial time $(1 + \varepsilon)$ -approximation.

Aiming to better understand GSP, in this paper we investigate the special case that all jobs are released at time 0. This case is still strongly NP-hard [20] and the currently best known approximation algorithm for it is a $(4 + \varepsilon)$ -approximation algorithm [18,23]⁴. As observed by Bansal and Verschae [8], this problem is a generalization of the covering-version of the well-studied Unsplittable Flow on a Path problem (UFP) [2,3,5,11,14,17]. The input of this problem consists of a path, each edge e having a demand u_e , and a set of tasks T . Each task i is specified by a start vertex s_i , an end vertex t_i , a size p_i , and a cost c_i . In the covering version, the goal is to select a subset of the tasks $T' \subseteq T$ which covers the demand profile, i.e., $\sum_{i \in T' \cap T_e} p_i \geq u_e$ where T_e denotes all tasks in T whose path uses e . The objective is to minimize the total cost $\sum_{i \in T'} c_i$.

This covering version of UFP has applications to resource allocation settings such as workforce and energy management, making it an interesting problem in its own right. For example, one can think of the tasks as representing time intervals when employees are available, and one aims at providing certain service level that changes over the day. UFP-cover is a generalization of the knapsack cover problem [12] and corresponds to instances of GSP without release dates where the cost function of each job attains only the values 0, some job-dependent value c_i , and ∞ . The best known approximation algorithm for UFP-cover is a 4-approximation [9,13], which essentially matches the best known result for GSP without release dates.

Our Contribution. In this paper we present several new approximation results for GSP without release dates and some of its special cases. First, we give a $(1 + \varepsilon)$ -approximation algorithm for the covering version of UFP with quasipolynomial running time. Our algorithm follows the high-level idea of the known QPTAS for the packing version [3]. Its key concept is to start with an edge in the middle and to consider the tasks using it. One divides these tasks into groups,

⁴ In [18] a *prima-dual* $(2 + \varepsilon)$ -approximation algorithm was claimed for this problem. However, there is a error in the argumentation: there are instances [23] where the algorithm constructs a dual solution whose value differs from the optimal integral solution by a factor of 4.

all tasks in a group having roughly the same size and cost, and guesses for each group an approximation of the capacity profile used by the tasks from that group. In the packing version, one can show that by slightly underestimating the true profile one still obtains almost the same profit as the optimum. For the covering version, a natural adjustment would be to use an approximate profile which *overestimates* the true profile. However, when using only a polynomial number of approximate profiles, it can happen that in the instance there are simply not enough tasks from a group available so that one can cover the overestimated profile which approximates the actual profile in the best possible way.

We remedy this problem in a maybe counterintuitive fashion. Instead of guessing an approximate upper bound of the true profile, we first guess a *lower* bound of it. Then we select tasks that cover this lower bound, and finally add a small number of “maximally long” additional tasks. Using this procedure, we cannot guarantee (instance-independently) how much our selected tasks exceed the guessed profile on each edge. However, we can guarantee that for the correctly guessed profile, we cover at least as much as the optimum and pay only slightly more. Together with the recursive framework from [3], we obtain a QPTAS. As an application, we use this algorithm to get a quasi-polynomial time $(e + \varepsilon)$ -approximation algorithm for GSP with uniform release dates, improving the approximation ratio of the best known polynomial time 4-approximation algorithm [18,23].

Moreover, we consider a different way to relax the problem. Rather than sacrificing a $1 + \varepsilon$ factor in the objective value, we present a polynomial time algorithm that computes a solution with *optimal* cost but requiring a speedup of $1 + \varepsilon$. Such a result can be easily obtained for job-*independent*, scalable cost functions using the PTAS in [22] (a cost function f is scalable if $f(ct) = \phi(c)f(t)$ for some suitable function ϕ and all $c, t \geq 0$). In our case, however, the cost functions of the jobs can be much more complicated and, even worse, they can be different for each job. Our algorithm first imposes some simplification on the solutions under consideration, at the cost of a $(1 + \varepsilon)$ -speedup. Then, we use a recently introduced technique to first guess a set of discrete intervals representing slots for large jobs and then use a linear program to simultaneously assign large jobs into these slots and small jobs into the remaining idle times [25].

An interesting open question is to design a (Q)PTAS for GSP without release dates. As a first step towards this goal, recently Megow and Verschae [22] presented a PTAS for minimizing the objective function $\sum_j w_j g(C_j)$ where each job j has a private weight w_j but the function g is identical for all jobs. In Section 4 we present a QPTAS for a generalization of this setting. Instead of only one function g for all jobs, we allow up to $(\log n)^{O(1)}$ such functions, each job using one of them, and we even allow the jobs to have up to $(\log n)^{O(1)}$ distinct release dates. Despite the fact that this setting is much more general, our algorithm is very clean and easy to analyze.

Related Work. As mentioned above, Bansal and Pruhs present a $O(\log \log P)$ -approximation algorithm for GSP [7]. Even for some well-studied special cases,

this is now the best known polynomial time approximation result. For instance, for the important weighted flow time objective, previously the best known approximation factors were $O(\log^2 P)$, $O(\log W)$ and $O(\log nP)$ [4,16], where P and W denote the ranges of the job processing times and weights, respectively. A QPTAS with running time $n^{O_\varepsilon(\log P \log W)}$ is also known [15]. For the objective of minimizing the weighted sum of completion times, PTASs are known, even for an arbitrary number of identical and a constant number of unrelated machines [1].

For the case of GSP with identical release dates, Bansal and Pruhs [7] give a 16-approximation algorithm. Later, Shmoys and Cheung claimed a primal-dual $(2+\varepsilon)$ -approximation algorithm [18]. However, an instance was later found where the algorithm constructs a dual solution which differs from the best integral solution by a factor 4 [23], suggesting that the primal-dual analysis can show only an approximation ratio of 4. On the other hand, Mestre and Verschae [23] showed that the local-ratio interpretation of that algorithm (recall the close relation between the primal-dual schema and the local-ratio technique [10]) is in fact a pseudopolynomial time 4-approximation, yielding a $(4+\varepsilon)$ -approximation in polynomial time.

As mentioned above, a special case of GSP with uniform release dates is a generalization for the covering version of Unsplittable Flow on a Path. For this special case, a 4-approximation algorithm is known [9,13]. The packing version is very well studied. After a series of papers on the problem and its special cases [5,11,14,17], the currently best known approximation results are a QPTAS [3] and a $(2+\varepsilon)$ -approximation in polynomial time [2].

2 Quasi-PTAS for UFP-Cover

In this section, we present a quasi-polynomial time $(1+\varepsilon)$ -approximation algorithm for the UFP-cover problem. Subsequently, we show how it can be used to obtain an approximation algorithm with approximation ratio $e+\varepsilon \approx 2.718+\varepsilon$ and quasi-polynomial running time for GSP without release dates. Throughout this section, we assume that the sizes of the tasks are quasi-polynomially bounded. Our algorithm follows the structure from the QPTAS for the packing version of Unsplittable Flow on a Path due to Bansal et al. [3]. First, we describe a recursive exact algorithm with exponential running time. Subsequently, we describe how to turn this routine into an algorithm with only quasi-polynomial running time and an approximation ratio of $1+\varepsilon$.

For computing the exact solution (in exponential time) one can use the following recursive algorithm: Given the path $G = (V, E)$, denote by e_M the edge in the middle of G and let T_M denote the tasks that use e_M . Our strategy is to “guess” which tasks in T_M are contained in OPT, the (unknown) optimal solution. Note that once these tasks are chosen, the remaining problem splits into the two independent subproblems given by the edges on the left and on the right of e_M , respectively, and the tasks whose paths are fully contained in them. Therefore, we enumerate all subsets of $T'_M \subseteq T_M$, denote by \mathcal{T}_M the resulting

set of sets. For each set $T'_M \in \mathcal{T}_M$ we recursively compute the optimal solution for the subpaths $\{e_1, \dots, e_{M-1}\}$ and $\{e_{M+1}, \dots, e_{|E|}\}$, subject to the tasks in T'_M being already chosen and that no more tasks from T_M are allowed to be chosen. The leaf subproblems are given when the path in the recursive call has only one edge. Since $|E| = O(n)$ this procedure has a recursion depth of $O(\log n)$ which is helpful when aiming at quasi-polynomial running time. However, since in each recursive step we try each set $T'_M \in \mathcal{T}_M$, the running time is exponential (even in one single step of the recursion). To remedy this issue, we will show that for any set \mathcal{T}_M appearing in the recursive procedure there is a set $\bar{\mathcal{T}}_M$ which is of small size and which approximates \mathcal{T}_M well. More precisely, we can compute $\bar{\mathcal{T}}_M$ in quasi-polynomial time (and it thus has only quasi-polynomial size) and there is a set $T_M^* \in \bar{\mathcal{T}}_M$ such that $c(T_M^*) \leq (1 + \varepsilon) \cdot c(T_M \cap \text{OPT})$ and T_M^* dominates $T_M \cap \text{OPT}$. For any set of tasks T' we write $c(T') := \sum_{i \in T'} c_i$, and for two sets of tasks T_1, T_2 , we say that T_1 *dominates* T_2 if $\sum_{i \in T_1 \cap T_e} d_i \geq \sum_{i \in T_2 \cap T_e} d_i$ for each edge e . We modify the above procedure such that we do recurse on sets in $\bar{\mathcal{T}}_M$ instead of \mathcal{T}_M . Since $\bar{\mathcal{T}}_M$ has quasi-polynomial size, $\bar{\mathcal{T}}_M$ contains the mentioned set T_M^* , and the recursion depth is $O(\log n)$, the resulting algorithm is a QPTAS. In the sequel, we describe the above algorithm in detail and show in particular how to obtain the set $\bar{\mathcal{T}}_M$.

2.1 Formal Description of the Algorithm

We use a binary search procedure to guess the optimal objective value B . First, we reject all tasks i whose cost is larger than B and select all tasks i whose cost is at most $\varepsilon B/n$. The latter cost at most $n \cdot \varepsilon B/n \leq \varepsilon B$ and thus only a factor $1 + \varepsilon$ in the approximation ratio. We update the demand profile accordingly.

We define a recursive procedure $\text{UFPcover}(E', T')$ which gets as input a subpath $E' \subseteq E$ of G and a set of already chosen tasks T' . Denote by \bar{T} the set of all tasks $i \in T \setminus T'$ such that the path of i uses only edges in E' . The output of $\text{UFPcover}(E', T')$ is a $(1 + \varepsilon)$ -approximation to the minimum cost solution for the subproblem of selecting a set of tasks $T'' \subseteq \bar{T}$ such that $T' \cup T''$ satisfy all demands of the edges in E' , i.e., $\sum_{i \in (T' \cup T'') \cap T_e} p_i \geq d_e$ for each edge $e \in E'$. Note that there might be no feasible solution for this subproblem in which case we output ∞ . Let e_M be the edge in the middle of E' , i.e., at most $|E'|/2$ edges are on the left and on the right of e_M , respectively. Denote by $T_M \subseteq \bar{T}$ all tasks in \bar{T} whose path uses e_M . As described above, the key is now to construct the set $\bar{\mathcal{T}}_M$ with the above properties. Given this set, we compute $\text{UFPcover}(E'_L, T' \cup T'_M)$ and $\text{UFPcover}(E'_R, T' \cup T'_M)$ for each set $T'_M \in \bar{\mathcal{T}}_M$, where E'_L and E'_R denote the subpaths of E' on the left and on the right of e_M , respectively. We output

$$\min_{T'_M \in \bar{\mathcal{T}}_M} c(T'_M) + \text{UFPcover}(E'_L, T' \cup T'_M) + \text{UFPcover}(E'_R, T' \cup T'_M).$$

For computing the set $\bar{\mathcal{T}}_M$, we first group the tasks in T_M into $(\log n)^{O(1)}$ many groups, all tasks in a group having roughly the same costs and sizes. Formally,

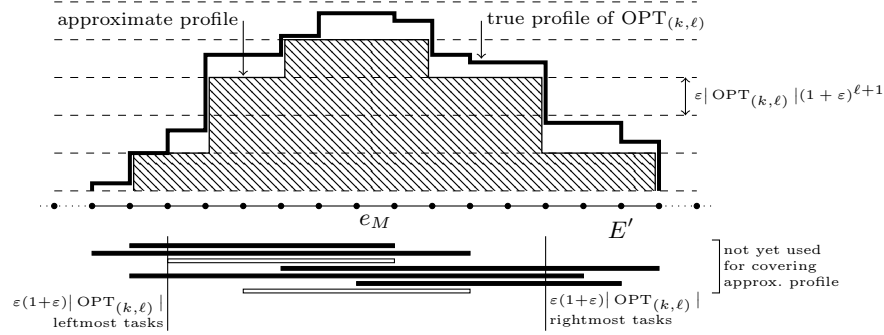


Fig. 1. Construction from Lemma 1.

for each pair (k, ℓ) , denoting (approximately) cost $(1 + \epsilon)^k$ and size $(1 + \epsilon)^\ell$, we define

$$T_{(k, \ell)} := \{i \in T_M : (1 + \epsilon)^k \leq c_i < (1 + \epsilon)^{k+1} \wedge (1 + \epsilon)^\ell \leq p_i < (1 + \epsilon)^{\ell+1}\}.$$

Since the sizes of the tasks are quasi-polynomially bounded and we preprocessed the weights of the tasks, we have $(\log n)^{O(1)}$ non-empty groups.

For each group $T_{(k, \ell)}$, we compute a set $\bar{T}_{(k, \ell)}$ containing at least one set which is not much more expensive than $\text{OPT}_{(k, \ell)} := \text{OPT} \cap T_{(k, \ell)}$ and which dominates $\text{OPT}_{(k, \ell)}$. To this end, observe that the sizes of the tasks in $\text{OPT}_{(k, \ell)}$ cover a certain profile (see Figure 1). Initially, we guess the number of tasks in $\text{OPT}_{(k, \ell)}$, and if $|\text{OPT}_{(k, \ell)}| \leq \frac{1}{\epsilon^2}$ then we simply enumerate all subsets of $T_{(k, \ell)}$ with at most $\frac{1}{\epsilon^2}$ tasks. Otherwise, we consider a polynomial number of profiles that are potential approximations of the true profile covered by $\text{OPT}_{(k, \ell)}$. To this end, we subdivide the (implicitly) guessed height of the true profile evenly into $\frac{1}{\epsilon}$ steps of uniform height, and we allow the approximate profiles to use only those heights while being monotonously increasing and decreasing before and after e_M , respectively (observe that also $\text{OPT}_{(k, \ell)}$ has this property since all its tasks use e_M). This leads to at most $n^{O(1/\epsilon)}$ different approximate profiles in total.

For each approximate profile we compute a set of tasks covering it using LP-rounding. The path of any task in $T_{(k, \ell)}$ contains the edge e_M , and hence, a task covering an edge e always covers all edges inbetween e and e_M as well. Thus, when formulating the problem as an LP, it suffices to introduce one constraint for the leftmost and one constraint for the rightmost edge of each height in the approximated profile. We compute an extreme point solution of the LP and round up each of the at most $\frac{2}{\epsilon}$ fractional variables. Since $|\text{OPT}_{(k, \ell)}| \geq \frac{1}{\epsilon^2}$ this increases the cost at most a factor $1 + O(\epsilon)$ compared to the cost of the LP.

It is clear that the LP has a solution if the approximate profile is dominated by the true profile. Among such approximate profiles, consider the one that is closest to the latter. On each edge it would be sufficient to add $O(\epsilon \cdot |\text{OPT}_{(k, \ell)}|)$ tasks from $T_{(k, \ell)}$ in order to close the remaining gap. This is due to our choice of the step size of the approximate profile and the fact that all tasks in $T_{(k, \ell)}$

have roughly the same size. To this end, from the not yet selected tasks in $T_{(k,\ell)}$ we add the $O(\varepsilon \cdot |\text{OPT}_{(k,\ell)}|)$ tasks with the leftmost start vertex and the $O(\varepsilon \cdot |\text{OPT}_{(k,\ell)}|)$ tasks with the rightmost end vertex (see Figure 1). This costs again at most an $O(\varepsilon)$ -fraction of the cost so far. As a result, on each edge e we have either selected $O(\varepsilon \cdot |\text{OPT}_{(k,\ell)}|)$ additional tasks using it, thus closing the remaining gap, or we have selected *all* tasks from $T_{(k,\ell)}$ using e . In either case, the selected tasks dominate the tasks in $\text{OPT}_{(k,\ell)}$, i.e., the true profile. The above procedure is described in detail in Appendix A.

Lemma 1. *Given a group $T_{(k,\ell)}$. There is a polynomial time algorithm which computes a set of task sets $\bar{T}_{(k,\ell)}$ which contains a set $T_{(k,\ell)}^* \in \bar{T}_{(k,\ell)}$ such that $c(T_{(k,\ell)}^*) \leq (1 + \varepsilon) \cdot c(\text{OPT}_{(k,\ell)})$ and $T_{(k,\ell)}^*$ dominates $\text{OPT}_{(k,\ell)}$.*

We define the set \bar{T}_M by taking all combinations of selecting exactly one set from the set $\bar{T}_{(k,\ell)}$ of each group $T_{(k,\ell)}$. Since there are $(\log n)^{O(1)}$ groups, by Lemma 1 the set \bar{T}_M has only quasi-polynomial size and it contains one set T_M^* which is a good approximation to $T_M \cap \text{OPT}$, i.e., the set T_M^* dominates $T_M \cap \text{OPT}$ and it is at most by a factor $1 + O(\varepsilon)$ more expensive. Now each node in the recursion tree has at most $n^{(\log n)^{O(1)}}$ children and, as argued above, the recursion depth is $O(\log n)$. Thus, a call to $\text{UFPcover}(E, \emptyset)$ has quasi-polynomial running time and yields a $(1 + O(\varepsilon))$ -approximation for the overall problem.

Theorem 1. *For any $\varepsilon > 0$ there is a quasi-polynomial $(1 + \varepsilon)$ -approximation algorithm for UFP-cover if the sizes of the tasks are in a quasi-polynomial range.*

Bansal and Pruhs [7] give a 4-approximation-preserving reduction from GSP with uniform release dates to UFP-cover using geometric rounding. Here we observe that if instead we use *randomized geometric rounding* [19], then one can obtain an e -approximation-preserving reduction. Together with our QPTAS for UFP-cover, we get the following result, whose proof we defer to Appendix A.

Theorem 2. *For any $\varepsilon > 0$ there is a quasi-polynomial time $(e + \varepsilon)$ -approximation algorithm for GSP with uniform release dates.*

3 General Cost Functions under Speedup

We present a polynomial time algorithm which computes a solution for an instance of GSP with uniform release dates whose cost is optimal and which is feasible if the machine runs with speed $1 + \varepsilon$ (rather than unit speed).

Let $1 > \varepsilon > 0$ be a constant and assume for simplicity that $\frac{1}{\varepsilon} \in \mathbb{N}$. For our algorithm, we first prove some properties that we can assume “at $1 + \varepsilon$ speedup”; by this, we mean that there is a schedule whose cost is at most the optimal cost (without enforcing these restricting properties) and which is feasible if we increase the speed of the machine by a factor $1 + \varepsilon$. Many statements are similar to properties that are used in [1] for constructing PTASs for the problem of minimizing the weighted sum of completion times.

For a given schedule denote by S_j and C_j the start and end times of job j in a given schedule (recall that we consider only non-preemptive schedules). We define $C_j^{(1+\varepsilon)}$ to be the smallest power of $1+\varepsilon$ which is not smaller than C_j , i.e., $C_j^{(1+\varepsilon)} := (1+\varepsilon)^{\lceil \log_{1+\varepsilon} C_j \rceil}$, and adjust the objective function as given in the next lemma. Also, we impose that jobs that are relatively large are not processed too early; formally, they do not run before $(1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1+\varepsilon) \rfloor}$ which is the largest power of $1+\varepsilon$ which is at most $\varepsilon/(1+\varepsilon) \cdot p_j$ (the speedup will compensate for the delay of the start time).

Lemma 2. *At $1+O(\varepsilon)$ speedup we can use the objective function $\sum_j f_j(C_j^{(1+\varepsilon)})$, instead of $\sum_j f_j(C_j)$, and assume $S_j \geq (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1+\varepsilon) \rfloor}$ for each job j .*

Next, we discretize the time axis into intervals of the form $I_t := [R_t, R_{t+1})$ where $R_t := (1+\varepsilon)^t$ for any integer t . Note that $|I_t| = \varepsilon \cdot R_t$. Following Lemma 2, to simplify the problem we want to assign an artificial release date to each job j . For each job j , we define $r(j) := (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1+\varepsilon) \rfloor}$. Lemma 2 implies then that we can assume $S_j \geq r(j)$ for each job j . Therefore, we interpret the value $r(j)$ as the release date of job j and from now on disallow to start job j before time $r(j)$.

In a given schedule, we call a job j *large* if $S_j \leq \frac{1}{\varepsilon^3} \cdot p_j$ and *small* otherwise. For the large jobs, we do not allow arbitrary starting times but we discretize the time axis such that each interval contains only a constant number of starting times for large jobs (for constant ε). For the small jobs, we do not want them to overlap over interval boundaries and we want that all small jobs scheduled in an interval I_t are scheduled during one (connected) subinterval $I_t^s \subseteq I_t$.

Lemma 3. *At $1+O(\varepsilon)$ speedup we can assume that*

- *each small job starting during an interval I_t finishes during I_t ,*
- *each interval I_t contains only $O(\frac{1}{\varepsilon^3})$ potential start points for large jobs, and*
- *for each interval I_t there is a time interval $I_t^s \subseteq I_t$, ranging from one potential start point for large jobs to another, which contains all small jobs scheduled in I_t and no large jobs.*

For the moment, let us assume that the processing times of the instance are polynomially bounded. We will give a generalization to arbitrary instances later.

Our strategy is the following: Since the processing times are bounded, the whole schedule finishes within $\log_{1+\varepsilon}(\sum_j p_j) \leq O_\varepsilon(\log n)$ intervals. Ideally, we would like to guess the placement of all large jobs in the schedule and then use a linear program to fill in the remaining small jobs. However, this would result in $n^{O_\varepsilon(\log n)}$ possibilities for the large jobs, which is quasi-polynomial but not polynomial. Instead, we only guess the *pattern* of large-job usage for each interval. A pattern P for an interval is a set of $O(\frac{1}{\varepsilon^3})$ integers which defines the start and end times of the large jobs which are executed during I_t . Note that such a job might start before I_t and/or end after I_t .

Proposition 1. *For each interval I_t there are only $N \in O_\varepsilon(1)$ many possible patterns. The value N is independent of t .*

We first guess all patterns for all intervals in parallel. Since there are only $O_\varepsilon(\log n)$ intervals, this yields only $N^{O_\varepsilon(\log n)} \in n^{O_\varepsilon(1)}$ possible combinations for all patterns for all intervals. Suppose now that we guessed the pattern corresponding to the optimal solution correctly. Next, we solve a linear program that in parallel assigns large jobs to the slots specified by the pattern, and also, it assigns small jobs into the remaining idle times on the intervals. Formally, we solve the following LP. We denote by Q the set of all slots for large jobs, $\text{size}(s)$ denotes the length of a slot s , $\text{begin}(s)$ its start time, and $t(s)$ denotes the index of the interval I_t that contains s . For each interval I_t denote by $\text{rem}(t)$ the remaining idle time for small jobs, and consider these idle times as slots for small jobs, which we refer to by their interval indices $I := \{1, \dots, \log_{1+\varepsilon}(\sum_j p_j)\}$. For each pair of slot $s \in Q$ and job $j \in J$, we introduce a variable $x_{s,j}$ corresponding to assigning j to s . Analogously, we use variables $y_{t,j}$ for the slots in I .

$$\min \sum_{j \in J} \left(\sum_{s \in Q} f_j(R_{t(s)+1}) \cdot x_{s,j} + \sum_{t \in I} f_j(R_{t+1}) \cdot y_{t,j} \right) \quad (1)$$

$$\sum_{s \in Q} x_{s,j} + \sum_{t \in I} y_{t,j} = 1 \quad \forall j \in J \quad (2)$$

$$\sum_{j \in J} x_{s,j} \leq 1 \quad \forall s \in Q \quad (3)$$

$$\sum_{j \in J} p_j \cdot y_{t,j} \leq \text{rem}(t) \quad \forall t \in I \quad (4)$$

$$x_{s,j} = 0 \quad \forall s \in Q, \forall j \in J : r(j) > \text{begin}(s) \vee p_j > \text{size}(s) \quad (5)$$

$$y_{t,j} = 0 \quad \forall t \in I, \forall j \in J : r(j) > R_t \vee p_j > \varepsilon \cdot |I_t|. \quad (6)$$

$$x_{s,j}, y_{t,j} \geq 0 \quad \forall s \in Q, \forall t \in I, \forall j \in J \quad (7)$$

Denote the above LP by sLP. It has polynomial size and thus we can solve it efficiently. Borrowing ideas from [24] we round it to a solution that is not more costly and which can be made feasible using additional speedup of $1 + \varepsilon$.

Lemma 4. *Given a fractional solution (x, y) to sLP. In polynomial time, we can compute a non-negative integral solution (x', y') whose cost is not larger than the cost of (x, y) and which fulfills the constraints (2), (3), (5), (6), (7) and*

$$\sum_{j \in J} p_j \cdot y_{t,j} \leq \text{rem}(t) + \varepsilon \cdot |I_t| \quad \forall t \in I. \quad (4a)$$

In particular, the cost of the computed solution is no more than the cost of the integral optimum and it is feasible under $1 + O(\varepsilon)$ speedup (accumulating all the speedups from the previous lemmas). We remark that the technique of guessing patterns and filling them in by a linear program was first used in [25].

For the general case, i.e., for arbitrary processing times, we first show that at $1 + \varepsilon$ speedup, we can assume that for each job j there are only $O(\log n)$ intervals between $r(j)$ (the artificial release date of j) and C_j . Then we devise a dynamic program which moves from left to right on the time axis and considers sets of $O(\log n)$ intervals at a time, using the above technique. See Appendix C for details.

Theorem 3. *Let $\varepsilon > 0$. There is a polynomial time algorithm for GSP with uniform release dates which computes a solution with optimal cost and which is feasible if the machine runs with speed $1 + \varepsilon$.*

4 Few Classes of Cost Functions

In this section, we study the following special case of GSP with release dates. We assume that each cost function f_j can be expressed as $f_j = w_j \cdot g_{u(j)}$ for a job-dependent weight w_j , k global functions g_1, \dots, g_k , and an assignment $u : J \rightarrow [k]$ of cost functions to jobs. We present a QPTAS for this problem, assuming that $k = (\log n)^{O(1)}$ and that the jobs have at most $(\log n)^{O(1)}$ distinct release dates. We assume that the job weights are in a quasi-polynomial range, i.e., we assume that there is an upper bound $W = 2^{(\log n)^{O(1)}}$ for the (integral) job weights.

In our algorithm, we first round the values of the functions g_i so that they attain only few values, $(\log n)^{O(1)}$ many. Then we guess the $(\log n)^{O(1)}/\varepsilon$ most expensive jobs and their costs. For the remaining problem, we use a linear program. Since we rounded the functions g_i , our LP is sparse, and by rounding an extreme point solution we increase the cost by at most an ε -fraction of the cost of the previously guessed jobs, which yields an $(1 + \varepsilon)$ -approximation overall.

Formally, we use a binary search framework to estimate the optimal value B . Having this estimate, we adjust the functions g_i such that each of them is a step function with at most $(\log n)^{O(1)}$ steps, all being powers of $1 + \varepsilon$ or 0.

Lemma 5. *At $1 + \varepsilon$ loss we can assume that for each $i \in [k]$ and each t it holds that $g_i(t)$ is either 0 or a power of $1 + \varepsilon$ in $[\frac{\varepsilon}{n} \cdot \frac{B}{W}, B)$.*

Our problem is in fact equivalent to assigning a due date d_j to each job (cf. [7]) such that the due dates are *feasible*, meaning that there is a preemptive schedule where every job finishes no later than its due date, and the objective being $\sum_j f_j(d_j)$. The following lemma characterizes when a set of due dates is feasible.

Lemma 6 ([7]). *Given a set of jobs and a set of due dates. The due dates are feasible if and only if for every interval $I = [r_j, d_{j'}]$ for any two jobs j, j' , the jobs in $X(I) := \{j : r_j \in I\}$ that are assigned a deadline after I have a total size of at least $\text{ex}(I) := \max(\sum_{j \in X(I)} p_j - |I|, 0)$. That is, $\sum_{j \in X(I): d_j > d_{j'}} p_j$ is at least $\text{ex}(I)$ for all intervals $I = [r_j, d_{j'}]$.*

Denote by D all points in time where at least one cost function g_i increases. It suffices to consider only those values as possible due dates.

Proposition 2. *There is an optimal due date assignment such that $d_j \in D$ for each job j .*

Denote by R the set of all release dates of the jobs. Recall that $|R| \leq (\log n)^{O(1)}$. We guess now the $|D| \cdot |R|/\varepsilon$ most expensive jobs of the optimal solution and their respective costs. Due to the rounding in Lemma 5 we have that $|D| \leq k \cdot$

$\log_{1+\varepsilon}(W \cdot n/\varepsilon) = (\log n)^{O(1)}$ and thus there are only $O(n^{|D| \cdot |R|/\varepsilon}) = n^{(\log n)^{O(1)}/\varepsilon}$ many guesses.

Suppose we guess this information correctly. Let J_E denote the guessed jobs and for each job $j \in J_E$ denote by d_j the latest time where it attains the guessed cost, i.e., its *due date*. Denote by c_{thres} the minimum cost of a job in J_E , according to the guessed costs. The remaining problem consists in assigning a due date $d_j \in D$ to each job $J \setminus J_E$ such that none of these jobs costs more than c_{thres} , all due dates together are feasible, and the overall cost is minimized. We express this as a linear program. In that LP, we have a variable $x_{j,t}$ for each pair of a job $j \in J \setminus J_E$ and a due date $t \in D$ such that j does not cost more than c_{thres} when finishing at time t . We add the constraint $\sum_{t \in D} x_{j,t} = 1$ for each job j , modeling that the job has a due date, and one constraint for each interval $[r, t]$ with $r \in R$ and $t \in D$ to model the condition given by Lemma 6. See Appendix D for the full LP.

In polynomial time, we compute an extreme point solution x^* for the LP. It has at most $|D| \cdot |R| + |J \setminus J_E|$ many non-zeros. Each job j needs at least one non-zero variable $x_{j,t}^*$, due to the constraint $\sum_{t \in D} x_{j,t} = 1$. Thus, there are at most $|D| \cdot |R|$ fractionally assigned jobs, i.e., jobs j having a variable $x_{j,t}^*$ with $0 < x_{j,t}^* < 1$. We define an integral solution by rounding x^* as follows: For each job j we set d_j to be the maximum value t such that $x_{j,t}^* > 0$. We round up at most $|D| \cdot |R|$ jobs and after the rounding, each of them costs at most c_{thres} . Hence, those jobs cost at most an ε -fraction of the cost of guessed jobs (J_E).

Lemma 7. *Denote by $c(x^*)$ the cost of the solution x^* . We have that*

$$\sum_{j \in J \setminus J_E} f_j(d_j) \leq c(x^*) + \varepsilon \cdot \sum_{j \in J_E} f_j(d_j).$$

Since $c(x^*) + \sum_{j \in J_E} f_j(d_j)$ is a lower bound on the optimum, we obtain a $(1 + \varepsilon)$ -approximation. As there are quasi-polynomially many guesses for the expensive jobs and the remainder can be done in polynomial time, we obtain a QPTAS.

Theorem 4. *There is a QPTAS for GSP, assuming that each cost function f_j can be expressed as $f_j = w_j \cdot g_{u(j)}$ for some job-dependent weight w_j and at most $k = (\log n)^{O(1)}$ global functions g_1, \dots, g_k , and that the jobs have at most $(\log n)^{O(1)}$ distinct release dates.*

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of FOCS 1999*, pages 32–44, 1999.
2. A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese. A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In *Proceedings of SODA 2014*, 2014.
3. N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of STOC 2006*, pages 721–729, 2006.
4. N. Bansal and K. Dhamdhere. Minimizing weighted flow time. *ACM T. Alg.*, 3(4), 2007.

5. N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *Proceedings of SODA 2009*, pages 702–709, 2009.
6. N. Bansal and K. Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. In *Proceedings of ESA 2012*, pages 145–156.
7. N. Bansal and K. Pruhs. The geometry of scheduling. In *Proceedings of FOCS 2010*, pages 407–414, 2010. See also <http://www.win.tue.nl/~nikhil/pubs/wflow-journ3.pdf>.
8. N. Bansal and J. Verschae. Personal communication.
9. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.
10. R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discrete Math.*, 19(3):762–797, 2005.
11. P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *Proceedings of FOCS 2011*, pages 47–56, 2011.
12. R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of SODA 2000*, pages 106–115, 2000.
13. V. T. Chakaravarthy, A. Kumar, S. Roy, and Yogish Sabharwal. Resource allocation for covering time varying demands. In *Proceedings of ESA 2011*, volume 6942 of *LNCS*, pages 543–554. 2011.
14. A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proceedings of APPROX 2002*, volume 2462 of *LNCS*, pages 51–66, 2002.
15. C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of STOC 2002*, pages 297–305, 2002.
16. C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of STOC 2001*, pages 84–93, 2001.
17. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM T. Alg.*, 3, 2007.
18. M. Cheung and D. B. Shmoys. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of APPROX 2011*, volume 6845 of *LNCS*, pages 135–146. 2011.
19. M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Inform. Comput.*, 131(1):63–79, 1996.
20. E. L. Lawler. A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness. *Ann. Discrete Math.*, 1:331–342, 1977.
21. L. Lovász and M. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 1986.
22. N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of ICALP 2013*, volume 7965 of *LNCS*, pages 745–756. 2013.
23. J. Mestre and J. Verschae. A 4-approximation for scheduling on a single machine with general cost function. <http://arxiv.org/abs/1403.0298>.
24. D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(1-3):461–474, 1993.
25. M. Sviridenko and A. Wiese. Approximating the configuration-LP for minimizing weighted sum of completion times on unrelated machines. In *Proceedings of IPCO 2013*, volume 7801 of *LNCS*, pages 387–398. 2013.

Appendix

A Omitted proofs from Section 2

In order to prove Lemma 1, we formally introduce the notion of a *profile*. A profile $Q : E' \rightarrow \mathbb{R}_{\geq 0}$ assigns a *height* $Q(e)$ to each edge $e \in E'$, and a profile Q *dominates* a profile Q' if $Q(e) \geq Q'(e)$ holds for all $e \in E'$. The profile Q_T *induced* by the tasks T is defined by the heights $Q_T(e) := \sum_{i \in T_e} p_i$, where T_e denotes all tasks in T whose path contains the edge e . Finally, a set of tasks T dominates a set of tasks T' if Q_T dominates $Q_{T'}$.

Lemma 1. Given a group $T_{(k,\ell)}$. There is a polynomial time algorithm which computes a set of task sets $\bar{T}_{(k,\ell)}$ which contains a set $T_{(k,\ell)}^* \in \bar{T}_{(k,\ell)}$ such that $c(T_{(k,\ell)}^*) \leq (1 + \varepsilon) \cdot c(\text{OPT}_{(k,\ell)})$ and $T_{(k,\ell)}^*$ dominates $\text{OPT}_{(k,\ell)}$.

Proof. In the first step, we guess the number of tasks in $\text{OPT}_{(k,\ell)} := T_{(k,\ell)} \cap \text{OPT}$. Abusing notation, we write $\text{OPT}_{(k,\ell)}$ also for the total cost of the tasks in $\text{OPT}_{(k,\ell)}$. If $|\text{OPT}_{(k,\ell)}|$ is smaller than $\frac{1}{\varepsilon^2}$ then we can guess an optimal set $\text{OPT}_{(k,\ell)}$. Otherwise, we will consider a polynomial number of certain approximate profiles one of which underestimates the unknown true profile induced by $\text{OPT}_{(k,\ell)}$ by at most $O(\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$. For each approximate profile we will compute a cover of cost no more than $1 + O(\varepsilon)$ the optimum, and in case of the profile being close to the true profile, we can extend this solution to a cover of the true profile by adding only $O(\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ more tasks.

Several arguments in the remaining proof are based on the structure of $T_{(k,\ell)}$ and the resulting structure of the true profile $Q_{\text{OPT}_{(k,\ell)}}$. Since all tasks in $T_{(k,\ell)}$ containing the edge e_M and spanning a subpath of E' , the height of the profile $Q_{\text{OPT}_{(k,\ell)}}$ is unimodal: It is non-decreasing until e_M and non-increasing after that; see Figure 1. In particular, a task that covers a certain edge e covers all edges in between e and e_M as well.

For the approximate profiles, we restrict to heights from

$$\mathcal{H} := \left\{ j \cdot \varepsilon \cdot |\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1} \mid j \in \{0, 1, \dots, \frac{1}{\varepsilon}\} \right\}.$$

Moreover, aiming to approximate the true profile, we only take into account profiles in which the edges have non-decreasing and non-increasing height before and after e_M on the path, respectively. Utilizing the natural ordering of the edges on the path, we formally define the set \mathcal{Q} of approximate profiles as follows

$$\mathcal{Q} := \left\{ Q \mid \begin{array}{l} Q(e) \in \mathcal{H} \quad \forall e \in E' \quad \wedge \quad Q(e) \leq Q(e') \quad \forall e < e' \leq e_M \\ \wedge \quad Q(e) \geq Q(e') \quad \forall e_M \leq e < e' \end{array} \right\}.$$

Since $|\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1}$ is an upper bound on the maximum height of $Q_{\text{OPT}_{(k,\ell)}}$, there is a profile $Q^* \in \mathcal{Q}$ which is dominated by $Q_{\text{OPT}_{(k,\ell)}}$ and for which the gap $Q_{\text{OPT}_{(k,\ell)}}(e) - Q(e)$ does not exceed $\varepsilon \cdot |\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{\ell+1}$ for all $e \in E'$. Observe that by construction, an approximate profile can have at

most $|\mathcal{H}|$ edges at which it jumps from one height to a larger one, and analogously, it can have at most $|\mathcal{H}|$ edges where it can jump down to some smaller height. Hence, \mathcal{Q} contains at most $n^{2|\mathcal{H}|} = n^{2/\varepsilon}$ profiles.

For each approximate profile $Q \in \mathcal{Q}$, we compute a cover based on LP rounding. To this end, we denote by $e_L(h)$ and $e_R(h)$ the first and last edge $e \in E'$ for which $Q(e) \geq h$, respectively. Note that by the structure of the paths of tasks in $T_{(k,\ell)}$, in fact every set of tasks covering $e_L(h)$ also covers all edges between e_M and $e_L(h)$ by at least the same amount, and analogously for $e_R(h)$. Regarding the LP-formulation, this allows us to only require a sufficient covering of the edges $e_L(h)$ and $e_R(h)$ rather than of all edges. Denoting by P_i the path of a task i , and by x_i the decision variable representing its selection for the cover, we formulate the LP as follows

$$\begin{aligned} \min \quad & \sum_{i \in T_{(k,\ell)}} c_i \cdot x_i \\ \sum_{i \in T_{(k,\ell)} : e_L(h) \in P_i} x_i \cdot p_i & \geq h & \forall h \in \mathcal{H} \\ \sum_{i \in T_{(k,\ell)} : e_R(h) \in P_i} x_i \cdot p_i & \geq h & \forall h \in \mathcal{H} \\ 0 & \leq x_i \leq 1 & \forall i \in T_{(k,\ell)}. \end{aligned}$$

If there exists a feasible solution to the LP, we round up all fractional values x_i^* (i.e., values $x_i^* \in (0, 1)$) of some optimal extreme point solution x^* , and we choose the corresponding tasks as a cover for Q and denote them by T^* . Since the LP has only $2|\mathcal{H}| = \frac{2}{\varepsilon}$ more constraints than variables, its optimal extreme point solutions contain at most $\frac{2}{\varepsilon}$ fractional variables. Hence, the additional cost incurred by the rounding does not exceed $\frac{2}{\varepsilon}(1 + \varepsilon)^{k+1}$, where the latter term is the maximum task cost in $T_{(k,\ell)}$. Let us assume for calculating the cost of the computed solution that $Q = Q^*$. Then, the cost of the selected tasks is at most

$$\begin{aligned} \sum_{i \in T_{(k,\ell)}} c_i \cdot x_i^* + \frac{2}{\varepsilon}(1 + \varepsilon)^{k+1} & \leq \text{OPT}_{(k,\ell)} + 2\varepsilon \cdot |\text{OPT}_{(k,\ell)}| \cdot (1 + \varepsilon)^{k+1} \\ & \leq (1 + 2\varepsilon(1 + \varepsilon)) \cdot \text{OPT}_{(k,\ell)}, \end{aligned}$$

where the first and second inequality follows from $|\text{OPT}_{(k,\ell)}| \geq \frac{1}{\varepsilon^2}$ and from the minimum task weight in $T_{(k,\ell)}$, respectively, and moreover, the first inequality uses that $Q = Q^*$ is dominated by $Q_{\text{OPT}_{(k,\ell)}}$.

After covering Q in the first step with T^* , in the second step, we extend this cover by additional edges $A^* \subseteq T_{(k,\ell)} \setminus T^*$. We define the set A^* to be the $\varepsilon(1 + \varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ tasks in $T_{(k,\ell)} \setminus T^*$ with the leftmost start vertices and the $\varepsilon(1 + \varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ tasks in $T_{(k,\ell)} \setminus T^*$ with the rightmost end vertices. We add $T^* \cup A^*$ to the set $\mathcal{T}_{(k,\ell)}$.

Assume that $Q = Q^*$. Then the above LP has a feasible solution and in particular the We claim that the computed tasks $T^* \cup A^*$ dominate $\text{OPT}_{(k,\ell)}$. Firstly, observe that any set of $\varepsilon(1 + \varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ tasks from $T_{(k,\ell)}$ has a

total size of at least the gap between two height steps from \mathcal{H} . Hence, if an edge e is covered by that many edges from A^* and $Q = Q^*$ then we know that $Q_{T^* \cup A^*}(e) \geq Q_{\text{OPT}_{(k,\ell)}}(e)$.

On the other hand, if an edge e is covered by less than $\varepsilon(1+\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ tasks from A^* , we know that there exists no further task in $T_{(k,\ell)} \setminus (T^* \cup A^*)$ whose path contains e . Otherwise, this would be a contradiction to the choice of the tasks A^* being the $\varepsilon(1+\varepsilon) \cdot |\text{OPT}_{(k,\ell)}|$ ones with the leftmost start and rightmost end vertices, respectively. Thus, since in this second case $T^* \cup A^*$ contains all tasks that cover e , we have that $Q_{T^* \cup A^*}(e) \geq Q_{\text{OPT}_{(k,\ell)}}(e)$.

Finally, the total cost of A^* does not exceed

$$2\varepsilon(1+\varepsilon) \cdot |\text{OPT}_{(k,\ell)}| \cdot (1+\varepsilon)^{k+1} \leq 2\varepsilon(1+\varepsilon)^2 \cdot \text{OPT}_{(k,\ell)}.$$

and thus the total cost of $T^* \cup A^*$ is upper-bounded by

$$(1 + 2\varepsilon(1+\varepsilon)(2+\varepsilon)) \cdot \text{OPT}_{(k,\ell)}.$$

We complete the proof by redefining ε appropriately. □

Theorem 2. For any $\varepsilon > 0$ there is a quasi-polynomial time $(e+\varepsilon)$ -approximation algorithm for GSP with uniform release dates.

Proof. The heart of the proof is an e -approximation-preserving reduction from GSP with uniform release dates to UFP-cover. Although here we develop a randomized algorithm, we note that the reduction can be de-randomized using standard techniques.

Given an instance of the scheduling problem we construct an instance of UFP-cover as follows. For ease of presentation, we take our path $G = (V, E)$ to have vertices $0, 1, \dots, P$; towards the end, we explain how to obtain an equivalent and more succinct instance. For each $i = 1, \dots, P$, edge $e = (i-1, i)$ has demand $u_e = P - i$.

The reduction has two parameters, $\gamma > 1$ and $\alpha \in [0, 1]$, which will be chosen later to minimize the approximation guarantee. For each job j , we define a sequence of times $t_0^j, t_1^j, t_2^j, \dots, t_k^j$ starting from 0 and ending with $P+1$ such that the cost of finishing a job in between two consecutive times differs by at most a factor of γ . Formally, $t_0^j = 0$, $t_k^j = P+1$ and t_i^j is the first time step such that $f(t_i^j) > \gamma^{i-1+\alpha}$. For each $i > 0$ such that $t_{i-1}^j < t_i^j$, we create a task covering the interval $[t_{i-1}^j, t_i^j - 1]$ having demand p_j and costing $f_j(t_i^j - 1)$.

Given a feasible solution of the UFP-cover instance, we claim that we can construct a feasible schedule of no greater cost. For each job j , we consider the right-most task chosen (we need to pick at least one task from each job to be feasible) in the UFP-cover solution and assign to j a due date equal to the right endpoint of the task. Notice that the cost of finishing the jobs by their due date equals the total cost of these right-most tasks. By the feasibility of the UFP-cover solution, it must be the case that for each time t , the total processing volume of jobs with a due date of t or great is at least $T - t + 1$. Therefore, scheduling the jobs according to earliest due date first, yields a schedule that meets all the due

date. Therefore, the cost of the schedule is at most the cost of the UFP-cover instance.

Conversely, given a feasible schedule, we claim that, if α is chosen uniformly at random and set $\gamma = e$, then there is a solution of the UFP-cover instance whose expected cost is at most e times more expensive than the cost of the schedule. For each job j , we pick all the tasks whose left endpoint is less than or equal to the completion time of j . It follows that the UFP-cover solution is feasible. Let $f_j(C_j)$ be the cost incurred by j . For a fixed α , let the most expensive task induced by j cost $f_j(C_j)\gamma^\beta$. Notice that β is also uniformly distributed in $[0, 1]$. The combined expected cost of all the tasks induced by j is therefore

$$\int_0^1 f_j(C_j) (\gamma^\beta + \gamma^{\beta-1} + \dots) d\beta = f_j(C_j) \frac{\gamma}{\ln \gamma},$$

which is minimum at $\gamma = e$. By linearity of expectation, we get that the total cost of the UFP-cover solution is at most an e factor larger than the cost of the schedule.

To de-randomize the reduction, and at the expense of adding another ε' to the approximation factor, one can discretize the random variable α , solve several instances, and return the one producing the best solution. Finally, we mention that it is not necessary to construct the full path from 0 to P . It is enough to keep the vertices where tasks start or end. Stretches where no task begins or end can be summarized by an edge having demand equal to the largest demand in that stretch.

Applying the e -approximation-preserving reduction and then running the $(1 + \varepsilon)$ -approximation of Theorem 2 finishes the proof. \square

B Omitted proofs from Section 3

In the following lemmas, we show different properties that we can assume at a speedup of $1 + \varepsilon$. In fact, each property requires to increase the speed by another factor of $1 + \varepsilon$. Compared to the initial unit speed, the final speed will be some power of $1 + \varepsilon$. Technically, we consolidate the resulting polynomial in ε to some $\varepsilon' = O(\varepsilon)$, achieving all properties of the lemmas at speed $1 + \varepsilon'$.

Lemma 2. At $1+O(\varepsilon)$ speedup we can use the objective function $\sum_j f_j(C_j^{(1+\varepsilon)})$, instead of $\sum_j f_j(C_j)$, and assume $S_j \geq (1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1+\varepsilon) \rfloor}$ for each job j .

Proof. Consider some job j with completion time C_j in an arbitrary schedule at unit speed. At speed $1 + \varepsilon$, time $C_j^{(1+\varepsilon)}$ corresponds to

$$(1 + \varepsilon)^{\left\lceil \log_{1+\varepsilon} \frac{C_j}{1+\varepsilon} \right\rceil} = (1 + \varepsilon)^{\lceil \log_{1+\varepsilon} C_j \rceil - 1} \leq C_j,$$

and hence, the ensued cost never exceeds the original cost.

Regarding the second point of the lemma, we observe that running a job j of processing time p_j at speed $1 + \varepsilon$ allows for an additional idle time of length $\varepsilon/(1 + \varepsilon) \cdot p_j$ compared to running it at unit speed. Hence, in case that $S_j < (1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1 + \varepsilon) \rfloor}$ we can set its start time to $(1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} \varepsilon \cdot p_j / (1 + \varepsilon) \rfloor}$ without exceeding its unit speed completion time.

Hence, we can make the assumptions of the lemma at a total speedup of $(1 + \varepsilon)^2$, which is $1 + O(\varepsilon)$ under our assumption that $\varepsilon < 1$, so the lemma follows. \square

Lemma 3 is restated in a slightly stronger way, the statement given here immediately implies the version in the main part of the paper.

Lemma 3. Let $I_{t,k} := [R_{t,k}, R_{t,k+1})$ where $R_{t,k} := (1 + k \cdot \frac{1}{4} \frac{\varepsilon^4}{1+\varepsilon}) R_t$ for $t \in \mathbb{N}$ and $k \in \{0, \dots, 4 \frac{1+\varepsilon}{\varepsilon^3}\}$.

- At $1 + \varepsilon$ speedup any small job starting during an interval I_t finishes in I_t .
- At $1 + \varepsilon$ speedup we can assume that each large job starts at some point in time $R_{t,k}$ and every interval $I_{t,k}$ is used by either only small jobs or by one large job or it is empty.
- For each interval I_t there is a time interval $I_{t,k,\ell} := [R_{t,k}, R_{t,\ell})$ with $0 \leq k \leq \ell \leq 4 \frac{1+\varepsilon}{\varepsilon^3}$ during which no large jobs are scheduled, and no small jobs are scheduled during $I_t \setminus I_{t,k,\ell}$.

Proof. Consider a small job that is started in I_t and that is completed in some later interval. By definition, its length is at most $\varepsilon^3 \cdot R_{t+1}$. At speed $1 + \varepsilon$, the interval I_t provides an additional idle time of length

$$\left(1 - \frac{1}{1+\varepsilon}\right) \varepsilon \cdot R_t = \frac{\varepsilon^2}{1+\varepsilon} \cdot R_t,$$

and the length of the small job reduces to at most $\varepsilon^3 \cdot R_t$. Since for sufficiently small ε it holds that $\frac{\varepsilon^2}{1+\varepsilon} \geq \varepsilon^3$, the small job can be scheduled during the idle time, and hence, it finishes in I_t .

Regarding the second point of the lemma, we observe that the length of a large job starting during I_t is at least $\varepsilon^3 \cdot R_t$ by definition. When running a large job at speed $1 + \varepsilon$, its processing time reduces by at least $\varepsilon^4/(1 + \varepsilon) \cdot R_t$ which equals four times the gap between two values $R_{t,k}$. If $I_{t,k}$ and $I_{t,\ell}$ are the first and last interval in I_t used by some large job j in a unit speed schedule then, at speed $1 + \varepsilon$, we can start j at time $R_{t,k+2}$, and it will finish no later than $R_{t,\ell-1}$ or it will finish in some later interval I_s , $s > t$. In case of job j finishing in I_t , the speedup allows us to assume j to block the interval $[R_{t,k+2}, R_{t,\ell-1})$, and we know that no other job is scheduled in this interval.

Otherwise, if j finishes in some later I_s , let $I_{s,m}$ be the subinterval of its completion. Since j is not necessarily large in I_s , its reduce in runtime due a speedup may only be marginal with respect to I_s . In $I_{s,m}$, the job j may be followed by a set of s -small jobs and a s -large job (both possibly not existing).

Analogously to the above argumentation, at a speedup of $1 + \varepsilon$, we can start the s -large job at time $R_{s,m+2}$, and the interval $I_{s,m+1}$ becomes empty. We use this interval to schedule the small jobs from $I_{s,m}$. This delays their start, however, they still finish in I_s which is sufficient: By the first part of Lemma 2, we can calculate the objective function as if every job finished at the next larger value R_r after its actual completion time, i.e., at the end of the interval I_r during which it finishes. Hence, within an interval I_r we can rearrange the intervals $I_{r,k}$ without changing the cost. This completes the proof of the second part of the lemma.

The proof of the third part is a straight-forward implication of its second part. By this we can assume that all small jobs are contained in intervals $I_{t,k}$ that contain no large jobs. Applying again the first part of Lemma 2, we can rearrange those intervals in such a way that they appear consecutively. \square

Lemma 4. Given a fractional solution (x, y) to sLP. In polynomial time, we can compute a non-negative integral solution (x', y') whose cost is not larger than the cost of (x, y) and which fulfills the constraints (2), (3), (5), (6), (7) and

$$\sum_{j \in J} p_j \cdot y_{t,j} \leq \text{rem}(t) + \varepsilon \cdot |I_t| \quad \forall t \in I. \quad (4a)$$

Proof. The proof follows the general idea of [24]. Given some fractional solution (x, y) to the sLP (2)–(7), we construct a fractional matching M in a bipartite graph $G = (V \cup W, E)$. For each job $j \in J$ and for each large slot $s \in Q$, we introduce vertices $v_j \in V$ and $w_s \in W$, respectively. Moreover, for each slot of small jobs $t \in I$, we add $k_t := \lceil \sum_{j \in J} y_{t,j} \rceil$ vertices $w_{t,1}, \dots, w_{t,k_t} \in W$. We introduce an edge $(v_j, w_s) \in E$ with cost $f_j(R_{t(s)+1})$ for all job-slot pairs for which $x_{s,j} > 0$, and we choose it to an extent of $x_{s,j}$ for M . Regarding the vertices $w_{t,1}, \dots, w_{t,k_t}$, we add edges in the following way. We first sort all jobs j with $y_{t,j} > 0$ in non-increasing order of their length p_j , and we assign them greedily to $w_{t,1}, \dots, w_{t,k_t}$; that is, we choose the first vertex $w_{t,\ell}$ which has not yet been assigned one unit of fractional jobs, we assign as much as possible of $y_{t,j}$ to it, and if necessary, we assign the remaining part to the next vertex $w_{t,\ell+1}$. Analogously to the above edges, we define the cost of an edge $(v_j, w_{t,\ell})$ to be $f_j(R_{t+1})$, and we add it fractionally to M according to the fraction $y_{t,\ell,j}$ of $y_{t,j}$ the job was assigned to $w_{t,\ell}$ by the greedy assignment. Note that $p_{t,\ell}^{\min} \geq p_{t,\ell+1}^{\max}$ for $\ell = 1, \dots, k_t - 1$ where $p_{t,\ell}^{\min}$ and $p_{t,\ell}^{\max}$ are the minimum and maximum length of all jobs (fractionally) assigned to $w_{t,\ell}$, respectively.

By construction, M is in fact a fractional matching, i.e., for every vertex $v_j \in V$ the set M contains edges whose chosen fractions add up to exactly 1. Moreover, the total cost of M equals the cost of the solution (x, y) . Due to standard matching theory, we know that there also exists an integral matching M' in G whose cost does not exceed the cost of M , and since G is bipartite, we can compute such a matching in polynomial time, see e.g., [21]. We translate M back into an integral solution (x', y') of the LP where we set $y_{t,j} = 1$ for every edge $(v_j, w_{t,\ell})$ in M . It remains to show that (x', y') satisfies (2), (3), (4a), (5),

(6) and (7). All constraints but (4a) are immediately satisfied by construction. In order to show that (4a) is satisfied observe that

$$\begin{aligned}
\sum_{j \in J} p_j \cdot y'_{t,j} &\leq \sum_{\ell=1}^{k_t} p_{t,\ell}^{\max} \leq p_{t,1}^{\max} + \sum_{\ell=2}^{k_t} p_{t,\ell}^{\max} \leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t-1} p_{t,\ell}^{\min} \\
&\leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t-1} \sum_{\substack{j \in J: \\ (v_j, w_{t,\ell}) \in E}} p_j \cdot y_{t,\ell,j} \leq \varepsilon \cdot |I_t| + \sum_{\ell=1}^{k_t} \sum_{\substack{j \in J: \\ (v_j, w_{t,\ell}) \in E}} p_j \cdot y_{t,\ell,j} \\
&= \varepsilon \cdot |I_t| + \sum_{j \in J} p_j \cdot y_{t,j} \leq \varepsilon \cdot |I_t| + \text{rem}(t),
\end{aligned}$$

where the third inequality follows from (6). \square

C Proof of Theorem 3 for general processing times

In this section, we provide the missing technical details which allow to generalize the proof of Theorem 3 from polynomially bounded processing times to general processing times.

Theorem 3. Let $\varepsilon > 0$. There is a polynomial time algorithm for GSP with uniform release dates which computes a solution with optimal cost and which is feasible if the machine runs with speed $1 + \varepsilon$.

We first prove that at $1 + \varepsilon$ speedup, we can assume that jobs “live” for at most $O(\log n)$ intervals, i.e., for each job j there are only $O(\log n)$ intervals between $r(j)$ (the artificial release date) and C_j . Then, we devise a dynamic program which moves on the time axis from left to right, considers blocks of $O(\log n)$ consecutive intervals at once and computes a schedule for them using the approach from Section 3.

Lemma 8. At $1 + \varepsilon$ speedup we can assume that $\frac{C_j}{r(j)} \leq q(n) := \frac{1}{\varepsilon^3} n + (1 + \varepsilon)^5$. Thus, $[r(j), C_j]$ is contained in at most $K \leq O_\varepsilon(\log n)$ intervals.

Proof. By using $1 + \varepsilon$ speedup we create an idle time of

$$|I_t| - \frac{1}{1+\varepsilon} \cdot |I_t| = \frac{\varepsilon}{1+\varepsilon} \cdot |I_t| = \frac{\varepsilon^2 (1+\varepsilon)^t}{1+\varepsilon} = \varepsilon^2 (1+\varepsilon)^{t-1}$$

in each interval I_t . Then, the idle time during the interval I_{t+s} with $s := \log_{1+\varepsilon} \left(\frac{n}{\varepsilon^3} \right) + 3$ can fit all jobs j with $r(j) \leq R_t$:

$$\varepsilon^2 (1+\varepsilon)^{t+s-1} = \varepsilon^2 (1+\varepsilon)^{t+\log_{1+\varepsilon}(n/\varepsilon^3)+2} = n \cdot \frac{1}{\varepsilon} (1+\varepsilon)^{t+2} \geq \sum_{j: r(j) \leq R_t} p_j,$$

where the last inequality is a consequence of Lemma 2 which implies in the case of $r(j) \leq R_t$

$$t \geq \left\lceil \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j \right) \right\rceil \geq \log_{1+\varepsilon} \left(\frac{\varepsilon}{1+\varepsilon} \cdot p_j \right) - 1 = \log_{1+\varepsilon} (\varepsilon \cdot p_j) - 2.$$

Since all jobs i with $r(i) \leq R_{t-1}$ can be assumed to be scheduled in the idle time of some earlier interval if necessary, we can assume $R_{t-1} < r(j) \leq R_t$, and hence,

$$\frac{C_j}{r(j)} \leq \frac{R_{t+s+1}}{R_{t-1}} = (1+\varepsilon)^{s+2} = \frac{1}{\varepsilon^3} \cdot n + (1+\varepsilon)^5.$$

In particular, it is sufficient to consider $s+2 = O_\varepsilon(\log n)$ intervals for processing a job. \square

Throughout the remainder of this section we denote by $K := \lceil \log_{1+\varepsilon}(q(n)) \rceil \in O_\varepsilon(\log n)$ where $q(n)$ is the polynomial from Lemma 8. Thus, K denotes the number of intervals between the time $r(j)$ and the completion time C_j of each job j .

If after the assumption of Lemma 8 there is a point in time s that will *not* schedule any job, i.e., there is no job j with $s \in [r(j), r(j) \cdot q(n))$, then we divide the instance into two independent pieces.

Proposition 3. *Without loss of generality we can assume that the union of all intervals $\bigcup_j [r(j), r(j) \cdot q(n))$ is a (connected) interval.*

For our dynamic program we subdivide the time axis into *blocks*. Each block B_i consists of the intervals $I_{i,K}, \dots, I_{(i+1) \cdot K-1}$. The idea is that in each iteration the DP schedules the jobs released during a block B_i in the intervals of block B_i and block B_{i+1} . So in the end, the intervals of each block B_{i+1} contain jobs released during B_i and B_{i+1} .

To separate the jobs from both blocks we prove the following lemma.

Lemma 9. *At $1 + \varepsilon$ speedup we can assume that during each interval I_t in a block B_{i+1} there are two subintervals $[a_t, b_t), [b_t, c_t) \subseteq I_t$ such that*

- during $[a_t, b_t)$ only small jobs from block B_i are scheduled and during $I_t \setminus [a_t, b_t)$ no small jobs from block B_i are scheduled,
- during $[b_t, c_t)$ only small jobs from block B_{i+1} are scheduled and during $I_t \setminus [b_t, c_t)$ no small jobs from block B_{i+1} are scheduled,
- a_t, b_t, c_t are of the form $(1+z \cdot \frac{\varepsilon^4}{4(1+\varepsilon)^2}) \cdot R_t$ for $x \in \mathbb{N}$ and $z \in \{0, 1, \dots, \frac{4(1+\varepsilon)^2}{\varepsilon^3}\}$ (so possibly $[a_t, b_t) = \emptyset$ or $[b_t, c_t) = \emptyset$).

Proof. Based on Lemma 3 we can assume that all small jobs that are started within I_t also finish in I_t ; moreover, they are processed in some interval $I_{t,k,\ell} \subseteq I_t$ which contains no large jobs (see Lemma 3 for the notation). By Lemma 8, the interval I_t can be assumed to contain only small jobs with release date in B_i and B_{i+1} , and by Lemma 2 we know that we can rearrange the jobs in I_t without changing the cost. Hence, for proving the lemma it is sufficient to show that we can split $I_{t,k,\ell}$ at some of the discrete points given in lemma, such that the small jobs released in B_i and B_{i+1} are scheduled before and after this point, respectively.

The interval $I_{t,k,\ell}$ starts at $(1 + \frac{1}{4} k \cdot \varepsilon^4 / (1 + \varepsilon)) \cdot R_t$ and its length is some integral multiple of $\frac{1}{4} \varepsilon^4 / (1 + \varepsilon) \cdot R_t$. At a speedup of $1 + \varepsilon$, the interval $I_{t,k,\ell}$

provides additional idle time of length at least $\frac{1}{4}\varepsilon^4/(1+\varepsilon)^2 \cdot R_t$ (if $I_{t,k,\ell}$ is not empty), which equals the step width of the discrete interval end points required in the lemma. Hence, by scheduling all small jobs released in B_i and B_{i+1} at the very beginning and very end of $I_{t,k,\ell}$, there must be point in time $s := (1 + z \cdot \frac{\varepsilon^4}{4(1+\varepsilon)^2}) \cdot R_t$ with $z \in \{0, 1, \dots, \frac{4(1+\varepsilon)^2}{\varepsilon^3}\}$ which lies in the idle interval between the two groups of small jobs. Finally, if setting a_t and c_t to the start and end of $I_{t,k,\ell}$, respectively, and if choosing $b_t := s$, we obtain intervals as claimed in the lemma. \square

Using Lemma 8 we devise a dynamic program. We work again with patterns for the intervals. Here a pattern for an interval I_t in a block B_i denotes $O(\varepsilon)$ integers which define

- the start and end times of the large jobs from B_{i-1} which are executed during I_t ,
- the start and end times of the large jobs from B_i which are executed during I_t ,
- a_t, b_t, c_t according to Lemma 9, implying slots for small jobs.

Denote by \bar{N} the number of possible patterns for an interval I_t according to this definition. Similarly as in Proposition 1 we have that $\bar{N} \in O_\varepsilon(1)$ and \bar{N} is independent of t .

Each dynamic programming cell is characterized by a tuple (B_i, P_i) where B_i is a block during which at least one job is released or during the block thereafter, and P_i denotes a pattern for all intervals of block B_i . For a pattern P_i , we denote by $Q_i(P_i)$ and $Q_{i-1}(P_i)$ the set of slots in B_i which are reserved for large jobs released in B_{i-1} and B_i , respectively. Moreover, for some interval I_t in B_i let $D_{i-1,t}(P_i)$ and $D_{i,t}(P_i)$ be the two slots for small jobs from B_{i-1} and B_i , respectively. The number of DP-cells is polynomially bounded as there are only n blocks during which at least one job is released and, as in Section 3, the number of patterns for a block is bounded by $\bar{N}^{O_\varepsilon(\log n)} \in n^{O_\varepsilon(1)}$.

The subproblem encoded in a cell (B_i, P_i) is to schedule all jobs j with $r(j) \geq I_{i,K}$ during $[R_{i,K}, \infty)$ while obeying the pattern P_i for the intervals $I_{i,K}, \dots, I_{(i+1),K-1}$. To solve this subproblem we first enumerate all possible patterns P_{i+1} for all intervals of block B_{i+1} . Suppose that we guessed the pattern P_{i+1} corresponding to the optimal solution of the subproblem given by the cell (B_i, P_i) . Like in Section 3 we solve the problem of scheduling the jobs of block B_i according to the patterns P_i and P_{i+1} by solving and rounding a linear program of the same type as sLP. Denote by $\text{opt}(B_i, P_i, P_{i+1})$ the optimal solution to this subproblem.

Lemma 10. *Given a DP-cell (B_i, P_i) and a pattern P_{i+1} . There is a polynomial time algorithm which computes a solution to the problem of scheduling all jobs released during B_i according to the patterns P_i, P_{i+1} which*

- *does not cost more than $\text{opt}(B_i, P_i, P_{i+1})$ and*
- *is feasible if during B_i and B_{i+1} the speed of the machine is increased by a factor $1 + \varepsilon$.*

Proof. The proof works analogously to the proof of Lemma 4. We formulate the following LP for (fractionally) solving the problem

$$\min \sum_{j \in J_i} \left(\sum_{\substack{s \in Q_i(P_i) \\ \cup Q_i(P_{i+1})}} f_j(R_{t(s)+1}) \cdot x_{s,j} + \sum_{t=i \cdot K}^{(i+2) \cdot K - 1} f_j(R_{t+1}) \cdot y_{t,j} \right) \quad (8)$$

$$\sum_{\substack{s \in Q_i(P_i) \\ \cup Q_i(P_{i+1})}} x_{s,j} + \sum_{t=i \cdot K}^{(i+2) \cdot K - 1} y_{t,j} = 1 \quad \forall j \in J_i \quad (9)$$

$$\sum_{j \in J_i} x_{s,j} \leq 1 \quad \forall s \in Q_i(P_i) \cup Q_i(P_{i+1}) \quad (10)$$

$$\sum_{j \in J_i} p_j \cdot y_{t,j} \leq |D_{i,t}(P_{i(t)})| \quad \forall t \in \{i \cdot K, \dots, (i+2) \cdot K - 1\} \quad (11)$$

$$x_{s,j} = 0 \quad \forall j \in J_i, \forall s \in Q : r(j) > \text{begin}(s) \vee p_j > \text{size}(s) \quad (12)$$

$$y_{t,j} = 0 \quad \forall t \in I, \forall j \in J_i : r(j) > R_t \vee p_j > \varepsilon \cdot |I_t| \quad (13)$$

$$x_{s,j}, y_{t,j} \geq 0 \quad \forall j \in J_i, \forall s \in Q_i(P_i) \cup Q_i(P_{i+1}), \quad \forall t \in \{i \cdot K, \dots, (i+2) \cdot K - 1\}. \quad (14)$$

where $J_i \subseteq J$ denotes the set of all jobs j with $r(j) \in B_i$, and $i(t)$ is the index of the block the interval I_t is contained in.

This LP has exactly the same structure as sLP (1)–(7) and hence, we obtain an analogous result to Lemma 4. This means that given a fractional solution (x, y) to the above LP, we can construct an integral solution (x', y') which is not more costly than (x, y) , and which fulfills all constraints (9)–(14) with (11) being replaced by the relaxed constraint

$$\sum_{j \in J_i} p_j \cdot y_{t,j} \leq |D_{i,t}(P_{i(t)})| + \varepsilon \cdot |I_t| \quad \forall t \in \{i \cdot K, \dots, (i+2) \cdot K - 1\}.$$

However, at speedup of $1 + \frac{\varepsilon}{1-\varepsilon} \in 1 + O(\varepsilon)$, an interval I_t provides an additional idle time of $\varepsilon \cdot |I_t|$ which allows for scheduling the potential job volume of by which we may exceed the capacity of the interval. Due to Lemma 2, this does not increase the cost of the schedule which concludes the proof. \square

By definition of the patterns, an optimal solution $\text{OPT}(B_{i+1}, P_{i+1})$ is independent of the patterns that have been chosen for earlier blocks. This is simply due to the separately reserved slots for jobs from different blocks within each pattern, i.e., a slot in B_{i+1} which is reserved for jobs from B_i cannot be used by jobs from B_{i+1} in any case. Hence, $\text{OPT}(B_i, P_i)$ decomposes into $\text{OPT}(B_{i+1}, P_{i+1})$ and $\text{opt}(B_i, P_i, P_{i+1})$ for a pattern $P_{i+1} \in \mathcal{P}_{i+1}$ which leads to the lowest cost, where \mathcal{P}_{i+1} denotes the set of all possible patterns for block B_{i+1} . Thus, formally it holds

$$\text{OPT}(B_i, P_i) = \min_{P_{i+1} \in \mathcal{P}_{i+1}} \text{OPT}(B_{i+1}, P_{i+1}) + \text{opt}(B_i, P_i, P_{i+1}). \quad (15)$$

This observation of an optimal substructure allows to easily formulate a DP. We interpret each cell (B_i, P_i) as a node in a graph, and we add an edge between cells (B_i, P_i) and (B_{i+1}, P_{i+1}) for all $P_i \in \mathcal{P}_i$ and $P_{i+1} \in \mathcal{P}_{i+1}$. For each triple B_i, P_i, P_{i+1} we compute a solution using Lemma 10, and we assign the cost of this solution to the edge $((B_i, P_i), (B_{i+1}, P_{i+1}))$. Due to (15), a minimum cost path in this $O(\text{poly}(n))$ size graph corresponds to a scheduling solution whose cost, at speed $1 + \varepsilon$, does not exceed the optimal cost at unit speed. This implies Theorem 3.

D Omitted proofs and LP from Section 4

Lemma 5. At $1 + \varepsilon$ loss we can assume that for each $i \in [k]$ and each t it holds that $g_i(t)$ is either 0 or a power of $1 + \varepsilon$ in $[\frac{\varepsilon}{n} \cdot \frac{B}{W}, B)$.

Proof. Denote by $g_i^{(1+\varepsilon)}$ the rounded cost functions for $i \in [k]$, i.e., formally we define

$$g_i^{(1+\varepsilon)}(t) := \begin{cases} \min \left\{ (1 + \varepsilon)^{\lceil \log_{1+\varepsilon}(g_i(t)) \rceil}, B \right\} & , \text{ if } g_i(t) > \frac{\varepsilon}{n} \cdot \frac{B}{W} \\ \frac{\varepsilon}{n} \cdot \frac{B}{W} & , \text{ if } 0 < g_i(t) \leq \frac{\varepsilon}{n} \cdot \frac{B}{W} \\ 0 & , \text{ if } g_i(t) = 0. \end{cases}$$

Consider some optimal schedule with completion time C_j for $j \in J$. Then it holds that

$$\begin{aligned} \sum_{j \in J} w_j \cdot g_{u(j)}^{(1+\varepsilon)}(C_j) &\leq \sum_{\substack{j \in J: 0 < g_{u(j)}(C_j) \\ \leq (\varepsilon \cdot B)/(n \cdot W)}} w_j \cdot \frac{\varepsilon \cdot B}{n \cdot W} + (1 + \varepsilon) \cdot \sum_{\substack{j \in J: g_{u(j)}(C_j) \\ > (\varepsilon \cdot B)/(n \cdot W)}} w_j \cdot g_{u(j)} \\ &\leq \varepsilon \cdot B \cdot \frac{1}{n} \sum_{\substack{j \in J: 0 < g_{u(j)}(C_j) \\ \leq (\varepsilon \cdot B)/(n \cdot W)}} \frac{w_j}{W} + (1 + \varepsilon) \cdot B \\ &\leq \varepsilon \cdot B + (1 + \varepsilon) \cdot B = (1 + 2\varepsilon) \cdot B. \end{aligned}$$

The lemma follows by redefining ε . \square

At this point, we give the full formulation of the LP described in short in Section 4. After guessing the $|D| \cdot |R|/\varepsilon$ most expensive jobs J_E , the solution to this LP is the basis for scheduling the remaining problem.

$$\min \sum_{j \in J \setminus J_E} \sum_{t \in D} x_{j,t} \cdot f_j(t) \quad (16)$$

$$\sum_{\substack{j \in (J \setminus J_E) \\ \cap X([r,t])}} \sum_{\substack{t' \in D: \\ t' > t}} p_j \cdot x_{j,t'} + \sum_{\substack{j \in J_E \cap X([r,t]): \\ d_j > t}} p_j \geq \text{ex}([r,t]) \quad \forall r \in R \quad \forall t \in D \quad (17)$$

$$\sum_{t \in D} x_{j,t} = 1 \quad \forall j \in J \setminus J_E \quad (18)$$

$$x_{j,t} = 0 \quad \forall j \in J \setminus J_E \quad \forall t \in D : \quad (19)$$

$$x_{j,t} \geq 0 \quad \forall j \in J \setminus J_E \quad \forall t \in D \quad (20)$$

Denote by x^* an optimal solution to this LP.

Lemma 7. Denote by $c(x^*)$ the cost of the solution x^* . We have that

$$\sum_{j \in J \setminus J_E} f_j(d_j) \leq c(x^*) + \varepsilon \cdot \sum_{j \in J_E} f_j(d_j).$$

Proof. Informally, the proof of this lemma has already been given in the main part of the paper. Here, we only add the missing formal step. We define an integral solution by simply rounding up the solution x^* . Formally, for each job j we define d_j to be the maximum value t such that $x_{j,t}^* > 0$. As observed in Section 4, the solution x^* has at most $|D| \cdot |R|$ fractional entries $0 < x_{j,t}^* < 1$, and hence, the rounding affects at most $|D| \cdot |R|$ variables whose corresponding cost $x_{j,t} \cdot f_j(t)$ do not exceed c_{thres} after the rounding. Thus, in the resulting schedule we have

$$\begin{aligned} \sum_{j \in J \setminus J_E} f_j(d_j) &\leq c(x^*) + |D| \cdot |R| \cdot c_{\text{thres}} \leq c(x^*) + \varepsilon \cdot \frac{1}{\varepsilon} \cdot |D| \cdot |R| \cdot c_{\text{thres}} \\ &\leq c(x^*) + \varepsilon \cdot \sum_{j \in J_E} f_j(d_j). \end{aligned}$$

This completes the proof. □